A Genetic Algorithm for the Constrained Forest Problem

Michael Laszlo[1], and Sumitra Mukherjee[2]
[1] Graduate School of Computer and Information Sciences, Nova Southeastern University, Fort Lauderdale, FL, USA.
Email: mjl@nova.edu
[2] Graduate School of Computer and Information Sciences, Nova Southeastern University, Fort Lauderdale, FL, USA.
Email: sumitra@nova.edu

***Abstract*— Given a graph with positive edge weights and a positive integer *m*, the Constrained Forest Problem (CFP) problem seeks a minimum-weight spanning subgraph each of whose components contains at least *m* vertices. Such a subgraph is called an *m*-forest. We present a genetic algorithm (GA) for CFP, which is NP-hard for *m*”4. Our GA evolves good spanning trees, as determined by the weight of the *m*-forest into which it can be partitioned by a simple greedy algorithm. Genetic operators include mutation, which replaces a spanning tree edge by a different edge that connects the same pair of components, and recombination, which combines the edge sets of two spanning trees to produce two new spanning trees. The GA discovers *m*-forests that are significantly better than those identified by best-known approximation algorithms for CFP, and identifies optimal solutions in small problems.**

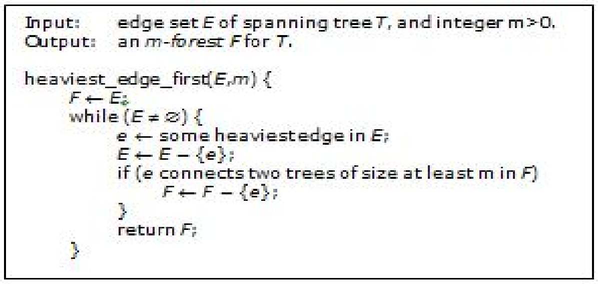
***Index Terms*— Genetic Algorithms, Spanning Forests, Constrained Forest Problem, Network Design Problems, Combinatorial Optimization.**

I. Introduction

Given a graph with positive edge weights, the Constrained Forest Problem (CFP) problem seeks a minimum-weight spanning subgraph each of whose components contains at least $m>1$ vertices. CFP belongs to a class of network design problems that have applications in such domains as VLSI layout and telecommunications network design. Network design problems assume this form: Given an edge-weighted graph $G=(V,E)$ and a *demand function* f: $2^V \to \{0,1\}$, find a minimum-weight set of edges that for all $S \subset V$, contains at least $f(S)$ edges from $\delta(S)$, where $\delta(S)$ contains exactly one endpoint in S. The CFP is modeled by the demand function $f(S)=1$ if and only if $0<|S|<m$. A forest is feasible if and only if $f(T)=0$ for every tree T. Imeliénska et al. [9] show that CFP is NP-hard for $m \geq 4$ and present a greedy heuristic for CFP that produces solutions within a factor of two of the optimum. Laszlo and Mukherjee [12] present a second greedy heuristic (which we refer to as *HEF* for *heaviest edge first*) that produces solutions at least as good as and often better than those produced by [9], and then locate both greedy heuristics within a family of heuristics all of which are 2-approximate for CFP [13]. Goemans and Williamson [6] treat CFP as a special case of network design problems with demand functions that are downwards monotone. They show their generalization of the greedy method of [9] to be 2-approximate. Goemans and Williamson [7] subsequently obtain a $(2-1/|V|)$-approximation method for CFP using a primal-dual approximation framework. Bar-Yehuda et al. [2] present a version of this method using the local ratio framework. Results on complexity and approximation of the CFP are presented in [3]. This paper presents a genetic algorithm (GA) to identify good solutions to CFP. The greedy HEF algorithm [12] achieves its 2-approximate performance guarantee by operating on the edges of the graph’s minimum spanning tree (MST). We show that there exist spanning trees that serve as better starting points, indeed, that there exist spanning trees that yield optimal solutions under HEF. Our GA is designed to evolve such spanning trees. Each individual in a population encodes a spanning tree of the input graph whose fitness reflects the quality of the m-forest partition that results under greedy HEF. Genetic operators include mutation, which replaces a spanning tree edge by a different edge that crosses the same cut, and recombination, which combines the edge sets of two spanning trees to produce two new spanning trees. Other studies have employed similar chromosome representations for GAs targeting network design problems (for example see [1], [11], [15], and [16]). Notably, our GA differs from other approaches in its choice of genotype-to-phenotype mapping and in its recombination operator. The paper is organized as follows. Section 2 motivates our use of the GA for this problem. Section 3 describes the GA. Section 4 presents examples on small data sets to illustrate our method. Section 5 presents and discusses the results of experiments designed to evaluate the GA’s efficacy. Section 6 concludes with some observations.

II. Motivation

A forest is called an *m-forest* if each of its trees contains at least m vertices for given $m>1$. The *heaviest-edge-first* (*HEF*) algorithm partitions a spanning tree $T=(V,E)$ into an m-forest spanning vertex set V [12]. This greedy algorithm iteratively processes spanning tree edges in decreasing weight order, and removes an edge if and only if it connects two components of size at least m (see Figure 1). The input is the edge set E and integer $m>1$; the output is a subset of E representing an m-forest spanning vertex set V. We use HEF(T) to denote the m-forest produced by applying HEF to spanning tree T where the value of m is assumed.



```
Input:    edge set $E$ of spanning tree $T$, and integer m>0.
Output:   an $m$-forest $F$ for $T$.

heaviest_edge_first($E$,$m$) {
     $F \leftarrow E$;
     while ($E \neq \varnothing$) {
          e $\leftarrow$ some heaviest edge in $E$;
          $E \leftarrow E - \{e\}$;
          if ($e$ connects two trees of size at least m in $F$)
               $F \leftarrow F - \{e\}$;
          }
          return $F$;
     }
```

Figure 1. Heaviest edge first (HEF) algorithm.

We can use the HEF algorithm in a 2-approximate greedy heuristic for CFP: First compute the MST of the input graph, and then apply HEF to the MST edges. Although the m-forests produced by this heuristic are no worse than twice optimal, in practice they may weigh close to this upper bound and it is not difficult to construct examples that do so. This suggests that there may exist spanning trees other than the MST which yield m-forests of less weight under HEF. In the remainder of this section, we show that for any input graph *G*, there exists a spanning tree *T* of *G* such that the m-forest HEF(*T*) is optimal. This motivates the use of a genetic algorithm to find such *HEF-optimal* trees in the search space of all spanning trees of G.. Let us say that a tree is *small* if it contains fewer than *m* vertices, and *large* otherwise. An edge *e* in tree *T* is *removable* if each of the two trees in the graph *T*–{*e*} is large. We refer to a tree or a forest as *irreducible* if it contains no removable edges (an irreducible forest comprises only irreducible trees). It is evident that the forests *F* produced by HEF are irreducible. Suppose to the contrary that *F* contains some tree *T* that is not irreducible, and let edge *eÎT* be some removable edge that connects two large subtrees C_1 and C_2. When HEF processed edge *e*, *e* must have connected two subtrees C'_1 and C'_2 containing C_1 and C_2 as subgraphs, implying that both C'_1 and C'_2 are large. Consequently, edge *e* would have been removed by HEF, contradicting our choice of edge *e*. We refer to a tree as *star-shaped* if it contains some vertex *v* such that every subtree to which *v* is connected by an edge is small. Vertex *v* is called a *center vertex* of the tree. It is shown in [14] that a tree is irreducible if and only if it is star-shaped. We use this result to prove the following theorem: *Theorem 1:* Given graph *G*, there exists some spanning tree *T* of *G* such that applying HEF to *T* yields an optimal m-forest. *Proof:* Given an optimal m-forest *F**, we construct a tree *T* such that HEF(*T*) yields *F**. Suppose that *F** contains the trees $T_1,\dots,T_k$. Let vertex v_i be some center vertex of tree T_i for i=1,…,k. First, add the edges of *F** to *T*. Then, for i=2,…k, add an edge connecting vertex v_i to v_1. To see that HEF(*T*)=*F**, let us refer to the edges of *F** as *original edges*, and the edges (v_i,v_1) that join two center vertices as *bridge edges*. Consider applying HEF to tree *T*, and let *e* be the current edge HEF is processing. There are two cases:

- If *e* is an original edge, suppose $e \in T_i$. Edge *e* connects two components one of which contains the center vertex v_i and the other of which, call it C_w, does not. Since tree T_i is irreducible, C_w is small, implying that edge *e* is retained by HEF.
- If *e* is a bridge edge, it connects some tree T_i to the component C that contains tree T_1. T_i is large because it belongs to an m-forest (*F**), and *C* is large because its subgraph T_1 is large; accordingly, edge *e* is removed by HEF. It follows that the HEF(*T*)=*F**. □

A spanning tree *T* for which HEF(*T*) is optimal is referred to as *HEF-optimal*. The tree construction of the previous proof can be generalized to show that multiple HEF-optimal trees may exist for a given input graph. The genetic algorithm described in the following section seeks an HEF-optimal tree in the search space of trees that span the input graph. Each chromosome represents a spanning tree *T*, and its fitness is a function of the weight of the m-forest HEF(*T*). The genetic operators are designed to move the search from the initial population, made up of copies of the input graph's MST, toward the HEF-optimal trees in the search space.

III. Genetic Algorithm

Genetic algorithms find wide use in combinatorial optimization problems [8]. A GA evolves a population of *chromosomes* (or *individuals*) that represent feasible solutions. Each chromosome stores a set of *genes* whose values collectively determine the chromosome's fitness. During each generation, genetic operators—selection, recombination, mutation, and replacement—are applied to the current population to produce the next generation. Selection selects individuals from the current population with probability proportional to their fitness values. Recombination operates on two selected chromosomes, swapping portions of each to produce two new chromosomes. Mutation alters the value of a gene. Replacement merges the current population and its offspring from which it selects the set of individuals to make up the next generation. Recombination focuses the search on promising neighborhoods of the feasible region while mutation widens the search to unexplored regions, thereby minimizing the risk of premature convergence to inferior solutions. The goal of a GA is to identify good solutions through this evolutionary process. We summarize the GA, then elaborate each phase, and then address time complexity.

```
Given a graph G, create the initial population of $m$
chromosomes.
For each of $N$ generations {
     Selection: Select $m$ parents based on the fitness, with
               replacement.
     Recombination: Pair parents randomly and perform
               recombination to produce offspring.
     Mutation: Mutate offspring.
     Replacement: Combine parents and offspring, and
               form the next generation of size $m$.
  }
```

Representation: Each chromosome represents a spanning tree of the input graph. The storage requirement for chromosomes is linear in the number of vertices in the graph. *Initialization*: The initial generation consists of μ copies of MSTs. The MST is likely to contribute much of the genetic material (i.e., edges) comprising an optimal solution. We can rely on mutation to supply new genetic material not present in the MST.

Fitness: The weight of the *m-forest* that results when HEF is applied to the set of edges represented by a chromosome is taken to be the chromosome's raw fitness. To obtain a scaled fitness value, we subtract the raw fitness from the total cost of the MST, and then linearly scale this value such that the maximum scaled fitness in the population is *f* times the average scaled fitness, where we use the value f=1.8 [8].

Selection: Using roulette-wheel sampling with replacement, we select μ chromosomes from the current population and place them in a mating buffer to serve as parents. The probability of a chromosome being selected in any trial is proportional to its scaled fitness value.

Recombination: Given two spanning trees with edge sets E_1 and E_2 that serve as parents, our recombination operator produces as offspring two spanning trees over the same set of edges $E_1 \cup E_2$. Each offspring retains the set of edges E_1E_2 common to both parents while exchanging some of the edges belonging to the symmetric difference $(E_1–E_2) \cup (E_2–E_1)$.

The recombination operator works as follows. First generate a uniform random number p_s between 0 and 0.5, and then create an edge set F containing a proportion p_s of the edges of $E_1–E_2$ selected at random. For each edge $e \in F$, remove e from E_1, thereby separating E_1 into two trees. Assign the label A to one of the trees and the label B to the other (the labels represent a partition or *cut* of the vertex set into two nonempty classes). Next, add edge e to edge set E_2, and then traverse the unique cycle in E_2 to which e belongs in order to discover some edge f, distinct from e, that connects an A vertex to a B vertex. Lastly, remove edge f from edge set E_2 and add it to E_1. By construction, E_1 and E_2 remain spanning trees, and edge f is drawn from the edge set $E_2–E_1$. Each of the two offspring resulting from recombination represents a spanning tree and hence encodes a feasible solution.

Mutation: With a small probability, each edge in a chromosome is removed and the two resulting components into which the spanning tree is partitioned are reconnected using a randomly selected edge. We bias the mutation operator to increase the likelihood of obtaining spanning trees with two desirable properties: First, spanning trees should be of low degree; vertices with high degree result in large star-shaped trees that restrict the number of edges HEF can remove. We facilitate this in our mutation operator by removing edges connected to vertices of high degree with higher probability. Secondly, since the optimal m-forest is likely to contain mostly edges of low weight, we use a mutation operator that is biased toward selecting light edges to replace mutated edges as follows: Randomly choose a fixed proportion of the vertices from each of the two components; call the subsets of selected vertices A and B. Select the lightest of the c edges connecting A and B to bridge the components.

Replacement: Replacement follows the elitist strategy. Specifically, the best $\mu/2$ parents and best $\mu/2$ offspring are merged to form the population for the next generation.

Complexity: Assume that graph G has $|E|$ edges and $|V|$ vertices. Cost of initialization depends on the cost of MST construction. A single MST of G can be constructed in $O(|E| + |V| \log |V|)$ time using Prim's algorithm and Fibonacci heaps. Fitness calculation is dominated by the cost of applying HEF to a spanning tree, which takes $O(m \cdot |E|)$ time. Given population size μ, formation of the offspring population via selection takes $O(\mu \log \mu)$ time, and elitist replacement also takes $O(\mu \log \mu)$ time. Recombination of two spanning trees with edge sets E_1 and E_2 adds an edge $e \in E_1–E_2$ to E_2, and then traverses the resulting cycle to discover an edge of $E_2–E_1$ to replace e. Such edge exchange may take $O(|V|)$ time since the cycle may be as large as $\Theta(|V|)$. Moreover, since $O(|V|)$ pairs of edges may be exchanged in this way, recombination takes $O(|V|^2)$ time in the worst-case sense. Mutation replaces an edge e by another edge that crosses the same cut as e. If each of the two components that e connects are size $\Theta(|V|)$, there may be as many as $O(|V|^2)$ edges to consider. However, because mutation considers only a fixed proportion of the candidate edges, the constant implicit in $O(|V|^2)$ can be made small.

IV. Example

We ran our GA on some small data sets to provide an illustration of the solutions it produces, and to compare the solutions to those obtained by the HEF method alone.

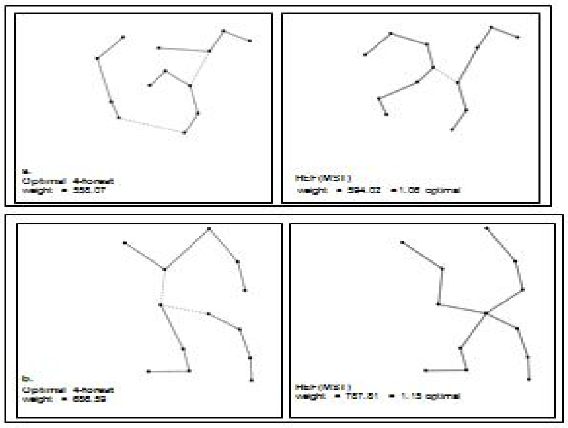


Figure 2. Optimal 4-forests with n=13.

Figure 2 shows optimal 4-forests obtained by our GA on two different complete 13-vertex graphs (panels a and b) using the Euclidean metric; it also presents the corresponding 4-forests produced by applying HEF to the graphs' MSTs. Edges removed from the spanning trees by HEF are shown as broken lines. Notice that the optimal solutions retain a subset of edges from the MSTs but also include non-MST edges. For n=13 and m=4 the number of feasible m-forests is 48,764 (see the appendix of [14] for a method to compute the number of feasible m-forests); the number of distinct spanning trees is over 1.79E+12. With an initial population of 20 MSTs, our GA found the optimal solution within 40 generations (thus exploring at most 800 spanning trees) in each of 20 consecutive runs. Each run took less than 2 seconds on a Pentium processor running Java bytecode at 2.5 Ghz.

V. Results

This section summarizes the results of our investigations for larger data sets. We used the TSP-1060 benchmark dataset taken from [18] to investigate the effectiveness of our recombination and mutation operators. The dataset contains 1060 two-dimensional points. Each point represents a vertex in a complete graph where edge weight is the Euclidean distance between pairs of vertices.

A. Recombination operator

Figure 3 presents the best solutions found by the GA over 800 generations in 6 different runs. The thin lines present the results of three runs where the recombination operator was applied with probability 0.8; the thick lines present results of runs without recombination. All other GA parameters are identical for both sets of runs. Notably, recombination consistently yields better quality solutions; over 30 GA runs, the mean decrease in weight in the best m-forests identified was 2.16 times greater when recombination was used.

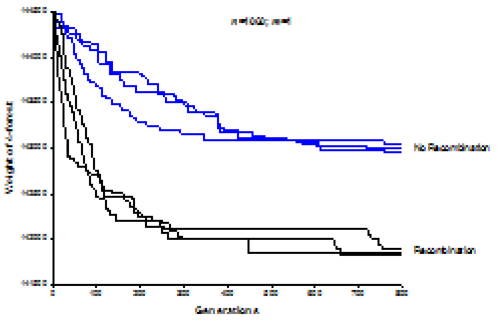


Figure 3. Best solutions found with recombination.

B. Biased mutation to reduce high degree vertices

We hypothesize that better quality solutions result when edges connected to vertices of high degree are selectively removed since high-degree vertices result in large star shaped trees and restrict the number of edges that HEF can remove. Results of our computational experiments to investigate this hypothesis are presented in Figure 4. We define *edge degree* to be the maximum of the degree of an edge's two endpoints. We bias mutation to favor removal of edges of high degree. Specifically, edges of degree two are mutated with probability p_{low}, edges of highest degree with probability p_{high}, and all remaining edges with probability p_{med}. In Figure 4, the thin line represents the best m-forest found over successive generations for the parameters p_{low}=0.001, p_{med}=0.002, and p_{high}=0.004. For comparison, the thick lines present results with the mutation probability set to constant $p_{low}=p_{med}=p_{high}$=0.001 and $p_{low}=p_{med}=p_{high}$=0.004, respectively. Results on 30 runs of the GA produced results consistent with our hypothesis. The mean decrease in weight in the best m-forests identified was 2.12 times greater when the mutation was biased as to reduce high degree vertices.

C. Biased mutation to introduce low-weight edges

We also conjecture that the optimal *m-forest* is likely to contain mostly edges of low weight and bias our mutation operator to replace mutated edges by edges of low weight. Figure 5 demonstrates the effect of selectively adding edges of low weight during mutation. The thin lines present the best solutions found over 800 generations on 3 runs of our GA under the following biased mutation scheme: The lightest edge bridging 40% randomly selected sample of vertices in each component resulting from the removal of an edge replaces the removed edge. The thick lines present the best solutions found when an arbitrary edge spanning the two components is selected. Notice that a bias towards introducing lighter edges results in better quality solutions.

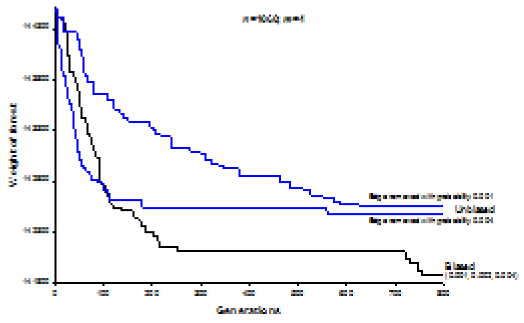


Figure 4. Biased mutation replaces edges connected to vertices of high degree.

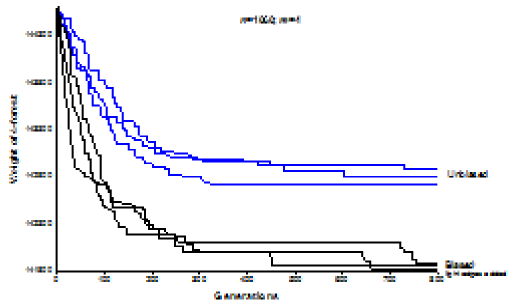


Figure 5. Biased mutation introduces low weight edges.

V. Conclusions

Our paper presents a genetic algorithm for finding good solutions for the NP-hard CFP problem. The GA exploits an existing heuristic for CFP in the mapping from genotype (spanning trees) to phenotype (m-forests), while outperforming the best-known heuristics when applied to large problems. When applied to small problems, the GA rapidly converges to an optimal solution. The representation of individuals and the choice of genetic operators that act on that representation are critical decisions in the design of a GA. We have chosen to represent individuals as edge sets that comprise spanning trees. This representation maps to an m-forest under the greedy HEF heuristic. Our primary genetic operators are recombination and mutation. Our recombination operator is a novel method for combining two individuals to form two new offspring composed of the same set of edges as their parents—the resulting offspring are feasible yet different from their parents. Mutation, which we explore in both biased and unbiased variants, transforms an individual into another by exchanging edges while ensuring the feasibility of the new individual. The representation of spanning trees as edge sets has been used to solve other intractable problems such as the degree-constrained minimum spanning tree [16] and minimum routing cost spanning tree [10] problems. Our GA's use of edge sets for CFP differs from prior use in two respects. First, because the genotype (spanning trees) and phenotype (m-forests) are of different type, the genotype-to-phenotype mapping is nontrivial. Second, the GA employs an innovative recombination operator on edge sets. These two novelties suggest that GAs of similar design might be useful for other optimization problems involving forests and other graph structures derivable from spanning trees.

REFERENCES

[1] Arnold B.C., N. Balakrishnan, and H.N. Nagaraja (1992), A First Course in Order Statistics, Wiley, New York.

[2] Bar-Yehuda, R., K. Bendel, A. Freund, and D. Rawitz (2004), "Local ratio: A unified framework for approximation algorithms", ACM Computing Surveys, 36, 4, 422–463.

[3] Bazgana C., Couëtouxa B. and Tuza Z. (2011). "Complexity and approximation of the Constrained Forest problem", Theoretical Computer Science, 412 (32), 22 4081-4091.

[4] Chou, H., G. Premkumar, and C.-H. Chu (2001), "Genetic algorithms for communications network design—an empirical study of the factors that influence performance," IEEE Transactions on Evolutionary Computation, 5, 236-249.

[5] Cordone, R., and F. Maffioli (2004), "On the complexity of graph tree partition problems", Discrete Applied Mathematics, 134, 51 – 65.

[6] Goemans, M., and D. Williamson (1994), "Approximating minimum-cost graph problems with spanning tree edges", Operations Research Letters, 16, 183–189.

[7] Goemans, M., and D. Williamson (1995), "A general approximation technique for constrained forest problems", SIAM Journal of Computing, 24, 2, 296–317.

[8] Goldberg, D. (1989). Genetic Algorithms in Search, Optimization and Machine Learning. Reading, MA: Addison-Wesley, 1989.

[9] Imieliñska, C., B. Kalantari, and L. Khachiyan, (1993) "A greedy heuristic for a minimum weight forest problem", Operations Research Letters,14, 65–71.

[10] Julstrom B.A., "The blob code is competitive with edge-sets in genetic algorithms for the minimum routing cost spanning tree problem", Proceedings of the Genetic and Evolutionary Computation Conference, GECCO-2005 (Hans-Georg Beyer et al., Editors), Volume 1, pp.585-590. New York: ACM Press.

[11] Knowles, J., and D. Corne (2000), "A new evolutionary approach to the degree constrained minimum spanning tree problem", IEEE Transactions on Evolutionary Computation, 4, 125-134.

[12] Laszlo, M., and S. Mukherjee, (2005) "Another greedy heuristic for the constrained forest problem", Operations Research Letters, 33(6), 629 – 633.

[13] Laszlo, M., and S. Mukherjee, (2006) "A class of heuristics for the constrained forest problem", Discrete Applied Mathematics, 154(1), 6-14.

[14] Laszlo, M., and S. Mukherjee, (2005), "Minimum spanning tree partitioning algorithm for microaggregation", IEEE Transactions on Knowledge and Data Engineering, 17(7), 902-911.

[15] Palmer, C., and A. Kershenbaum, (1994), "Representing trees in genetic algorithms", Proc. 1st IEEE Conf. Evolutionary Computation, 379-384.

[16] Raidl, G. R., and B. A. Julstrom. (2003) "Edge sets: an effective evolutionary coding of spanning trees", IEEE Transactions on Evolutionary Computation, 7(3), 225-239.

[17] Raidl, G. R., and B. A. Julstrom (2002) "Initialization is robust in evolutionary algorithms that encode spanning trees as sets of edges", Proceedings of the 2002 ACM symposium on Applied computing, Madrid, Spain, 547-552.

[18] Reinelt, G., (1991) "TSP-LIB A traveling salesman library", ORSA Journal of Computing, 3, 376-384.